LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-666558

Predicting Optimal Power Allocation for CPU and DRAM Domains

A. Tiwari, M. Schulz, L. Carrington

January 26, 2015

The 16th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing
Hyderabad, India
May 29, 2015 through May 29, 2015

Disclaimer

Predicting Optimal Power Allocation for CPU and DRAM Domains

Ananta Tiwari†, Martin Schulz‡, Laura Carrington†

†Performance Modeling and Characterization Laboratory
San Diego Supercomputer Center
San Diego, California, USA
{*tiwari, lcarring*}*@sdsc.edu*

‡Lawrence Livermore National Laboratory (LLNL)
Livermore, CA
schulzm@llnl.gov

***Abstract*—Constraints imposed by power delivery and costs will be key design impediments to the development of next generation High-Performance Computing (HPC) systems. To remedy these impediments, solutions that impose power bounds (or caps) on over-provisioned computing systems to remain within the physical (and financial) power limits have been proposed. Uninformed power capping can significantly impact performance and its success depends largely on how intelligently a given power budget is allocated across various subsystems of the computing nodes. Since different computations put vastly different demands on various system components, those variations in the demands must be taken into consideration while making power allocation decisions to lessen performance degradation. Given a target power bound, a model-based methodology presented in this paper, which takes computation-specific properties into account, guides power allocations for CPU and DRAM domains to maximize performance. Our methodology is accurate and can predict the performance impacts of the power capping allocation schemes for different types of computations from real applications with absolute mean error of less than 6%.**

I. Introduction

Utility costs and constraints on power delivery are limiting the expansion of large scale High-Performance Computing (HPC) systems. Future HPC systems will most likely be over-provisioned and power constrained to be operated within strict power limits [9], [17], [22]; i.e., not all components of the system can run at maximum power draw and therefore maximum performance. For these classes of systems, the objective of performance optimization, which has traditionally been to minimize time-to-solution, will morph into a more complicated objective – maximizing performance under a power budget, where power budget for a given system will depend on the physical (in terms of power delivery and cooling infrastructure) and financial considerations.

To operate power limited systems efficiently, decisions will have to be made to allocate the available power budget to different subcomponents (CPU, uncore and memory) within a compute node[1]. Some of these allocation decisions could lead to operating system components at reduced power levels, thereby degrading their performance. In order to lessen the impact on application execution time, power allocation decisions will have to be *application-aware*; i.e., power should be capped on component(s) that have the least impact on a given application's execution time. For example, power capping CPU subsystem during a computational phase that is memory bound will have less impact on application performance than power capping DRAM subsystem during the same computational phase.

Determining the extent to which different types of computations are sensitive to reduced power caps on CPU and DRAM subsystems is, therefore, the prerequisite in the development of optimal power capping strategies. Towards that end, this paper presents a systematic computation-aware methodology to understand and predict how sensitive a given computation is to different levels of power capping on CPU and DRAM subsystems in order to determine the optimal power

[1]We note that power limits could also be placed on other components such as Network and I/O. This work focuses at intra-compute-node power allocations to processor and DRAM subsystems.

distribution to components under a given power budget. In addition to different power capping levels imposed on CPU and DRAM, our predictive machine learning models utilize the application's computational properties as predictors – for example, arithmetic intensity, memory access patterns, cache hit rates and data dependencies. The models can be used to explore the performance sensitivity of different computations when different power caps are imposed simultaneously on CPU and DRAM domains. The models can also be used to select the best implementation of a computation, from among a set of source code variants that use different code optimizations, given a stipulated power budget.

In particular, this paper makes the following contributions:

1) We present models that are highly accurate in predicting the performance sensitivity of various HPC computations across a range of simultaneous power caps on CPU and DRAM domains.
2) We illustrate the use of computational characterization in developing accurate predictive models for performance degradations introduced by power limits and discuss the relative importance of different application properties in driving the observed degradations.
3) We demonstrate the effectiveness of our models in predicting the sensitivity to power capping of real applications and in choosing the performance optimal implementation variants (within an autotuning workflow) for a given power bound.

This paper is organized as follows. The next section describes related research, which is followed by the presentation of a study that motivates using a methodology that is based on application-specific attributes for predicting the optimal power capping levels for CPU and DRAM (Section III). The same section also describes the tools that we rely on to gather application characterizations and the modeling technique that we use in this paper. Section IV presents modeling results along with model evaluation results on mini applications. Finally, Section V provides concluding remarks.

II. Related Work

Power capping has recently received its fair share of attention from researchers in power-aware HPC. Rountree et al. [20] investigate the performance impact of power capping the Package (PKG) domain but not the DRAM domain for a SandyBridge system. Fukazawa et al. [12] investigate the performance variation seen in different sections of a large scale Magneto hydrodynamic (MHD) simulation code subjected to power capping. Patki et al. [17] look at how applications behave under RAPL power capping on the PKG domain and identify how application characteristics affect the performance optimal configuration (e.g., number nodes, cores per node, and power limit per node) under a power limit. Sarood et al. [22] utilize application profiles and a simple interpolation scheme to optimize performance of an application under a power budget by using RAPL to distribute the power between the CPU and memory and adding additional nodes. The interpolation scheme requires measuring the application at different power distributions under strong scaling to develop their application-specific models. Our work is distinct in that our machine-learning models are trained using computational characteristics from a set of benchmarks to predict the effect of CPU and DRAM power capping on any given application at fine granularities.

Etinski et al. [10] have worked on power reduction via DVFS to schedule jobs with a given power budget. Li et al. [15] also looked at scheduling within a power budget and utilized IPC models to determine the most energy efficient frequency and concurrency to be used in a hybrid programmed application. Goel et al. [13] use on-chip temperature measurements along with hardware counters to model power draw for the core to drive power-aware scheduling and frequency scaling to compute within a power budget. Our work differs in that we do not use hardware counters and our models enable "shifting" of power between the components to compute within a power limit. Similar to this work, power models for CPU and DRAM were developed to investigate energy efficient code variants for computational kernels [25]. The models in that work were application specific where as in this work the models are general and based on computational characteristics. In our earlier work [24], we presented a modeling framework that could be used to predict how different computations (and phases) within an application would behave under a given reduced per core memory bandwidth.

III. Motivation and Methodology

This section describes the motivation for the research presented in this paper and our methodology to predict DRAM and CPU power capping settings that maximize performance under a given power budget.

A. Motivation

The key insight, on which our methodology is based, is that different computations and even different implementations of the same computation will have different performance behavior in power-capped environments. We illustrate this insight with simple examples presented in Figure 1 and Figure 2. We took a set of diverse HPC computations (more details on what these computations consist of appear in Section IV) and ran them on our experimental testbed, which is a dual-socket SandyBridge-based system (also described in Section IV), with no power limits (base case). For each of the computations, we then ran two sets of experiments – one by imposing power limits on the DRAM domain and the other by imposing limits in the CPU domain. The limits for each domain are chosen based on the power draw that a given computation draws when there is no power limit imposed on the system. Figure 1 shows the results for the DRAM domain and Figure 2 shows the results for the CPU domain. The x-axis shows the power capping levels in terms of the proportion of the power drawn in the base case; e.g., a level of 0.62 indicates a power limit of 62% of the power drawn in the base case (i.e., a 38% reduction in power). Thus, the closer a point is to the y-axis the larger the power reduction. The y-axis shows the performance degradation with respect to the base case, which ranges from around 1.17X-5X.

The impact that reducing power cap in the DRAM domain has on performance is fairly extensive. If we consider a power level of 0.62 on the DRAM domain (see Figure 1), the performance degradations can be in the range of $2.5X$ to $5X$. While the range is not as extensive in the CPU power reduction, the range is still very significant. Where in the performance degradation range a computation falls really depends on its characteristics and determining what characteristics are key indicators of this degradation are requirements for any methodology that aims to maximize performance under given power budget. The next section describes the tools that we use to extract these key performance metrics of the computations within HPC applications.

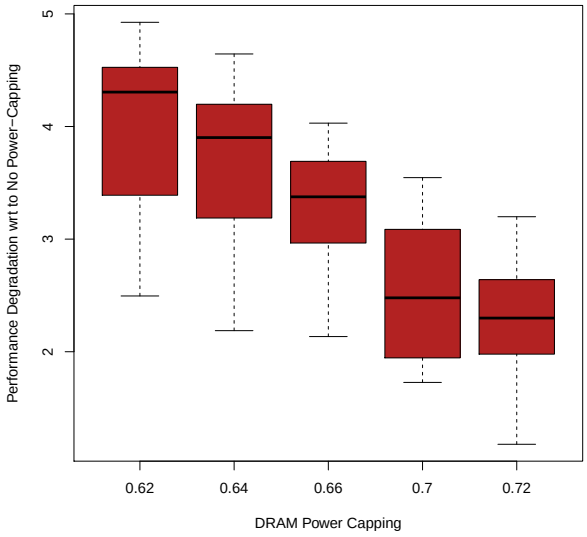

Fig. 1. The range of slowdowns for different dram power reductions, while imposing no power limit on the CPU domain. X-axis shows the power capping levels, where 0.62 denotes a 38% reduction in power.

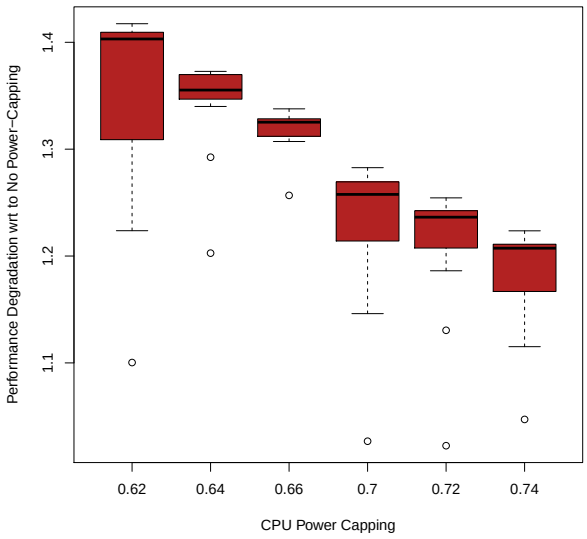

Fig. 2. The range of slowdowns for different CPU power reductions, while imposing no power cap on the DRAM domain. X-axis shows the power capping levels, where 0.62 denotes a 38% reduction in power.

B. Computation Characterization

In order to develop models that capture a computation's sensitivity to different power caps placed on the CPU and DRAM domains, we need to first capture low-level details of how an application interacts with and exercises the underlying hardware subcomponents. To do that, we leverage a suite of application analysis tools developed on top of PEBIL [14], which is a binary instrumentation toolkit for x86/Linux. PEBIL takes as input an application binary; it disassembles the binary, analyzes it, and can insert instrumentation(s). Tools written on top

of PEBIL can perform static and dynamic analysis on the application binary.

The static analysis tool written on top of PEBIL produces information about the approximate structure of the program (e.g., functions and loops) and the operations that occur within those structures. The tool also records type and size of classes of operations (e.g., memory and floating operations) that are within those control structures. The static analysis tool records the average size of memory operands in each block and measures the number of instructions between register or memory definitions and their usage (i.e., data dependencies).

To gather detailed information about data movement within an application, the memory characterization tool, a dynamic analysis tool written on top of PEBIL, instruments every memory access in the application and pipes the address stream to be processed on-the-fly to estimate various data motion related metrics (e.g., reuse distance, working set sizes of different loops and cache hit rates). Another dynamic analysis tool keeps visit count information for the application's control units (e.g., basic block visit counts). Visit count information when combined with the static instruction mix information gives detailed information on the instruction make-up of the application.

Information collected by static and dynamic tools can be combined to identify computational phases within large applications [26] and construct what we refer to as computation signature for each of the identified phases. Computation signature consist of the operations required by the application in the form of instruction mix and counts, data locality properties, metrics that capture the application's interaction with the memory subsystem such as cache hit rates, load and store operations, etc.

C. Modeling: Problem Formulation and Technique

As shown in Equation 1, our goal is to develop a function f, whose inputs are a set of hardware parameters (power caps) and application characterization metrics $x_1 \ \cdots \ x_n$ and whose output y is some measure of the effect on performance on the computation.

$$y = f(x_1 \ \cdots \ x_n) \tag{1}$$

In this work, we use Cubist [21], which is a rule-based machine learning technique, to estimate y in Equation 1. A Cubist model consists of a tree of linear regression models; predictions are based on the linear model found at the terminal/leaf node of the tree. The path to the leaf or the selection of the terminal node is based on the rules at intermediate nodes that are also based on linear models [18]. Unlike other forms of tree based methods, such as gradient boosting, Cubist models are easy to interpret. Cubist models also have the capability to encapsulate complex non-linear relationships between predictor (or input) variables. Intermediate rules, for example, could identify and separate differences in performance differences brought about by power capping for working set sizes that cross cache-hierarchy boundaries (e.g., L2 vs. L3 resident working set sizes).

To avoid over-fitting, we employ two techniques. First, we divide the empirical samples into non-overlapping training and test sets. The model is trained on the training subset and validated on the test set. Second, we use 10-fold cross-validation to produce the models. In k-fold cross-validation (in our case, $k = 10$), the training dataset is randomly partitioned into k subsets of approximately equal size. k different models are then constructed, each using $(k-1)$ of the k partitions as training input so that 1 of the k sets can be used for model validation. Each of the k models is then validated against the validation set and the model that yields the minimum error is selected.

IV. Results

This section starts by describing the experimental system that we use to collect our training data. We then describe a set of benchmarks that we use to gather training and test data, which is followed by the discussion of the accuracy of trained models on test dataset as well as on mini applications.

A. Experimental testbed

Our test system consists of a dual-socket server with two 8-core 2.9GHz Intel Xeon E5-2690 (SandyBridge) processors and 64GB of DDR3 memory. Simultaneous Multithreading (SMT) and Turbo-boost are disabled on the system.

The RAPL (Running Average Power Limit) [8] interface available on recent Intel[2] processors enables the collection of (modeled) power measurements for CPU and DRAM subsystems (RAPL documentation refers to these subsystems as "domains"). RAPL also allows users to set power limits on these domains and the underlying hardware infrastructure enforces these power limits. On our test Sandybridge-based system, RAPL exposes four

[2]While our focus is on using Intel processors, other chip manufacturers have also enabled power capping in their processors (e.g. AMD [3] and IBM [6], [5]). The methodology presented in the paper should be applicable to those architectures as well.

power planes – Package (PKG), Power Plane 0 (PP0), Power Plane 1 (PP1) and DRAM. We use Intel's Power Governor tool [1] to enforce power caps on the PKG and DRAM domains. We also rely on the Power Governor tool for DRAM and CPU power measurements. We measure 5 power samples every second, which, in our experience, incurs only negligible overhead.

B. Model Training Computations

To train models that capture how the performance of various types of computations changes when subjected to different power capping levels on the CPU and DRAM domains, we utilize a diverse set of computational kernels that are highly prevalent in HPC applications. The benchmarks, which are partly derived from PolyBench [19] and SPAPT [4] suites, consist of kernels derived from different computational domains – dense linear algebra (e.g., matrix-matrix multiplication and matrix-vector multiplication), stencil computations, linear algebra solvers (e.g., LU decomposition), etc. In addition, to expand and enrich the training space, for some of the kernels, we generate implementation variants (code-variants) using two source-to-source compiler transformation tools – Orio [16] and CHiLL [7]. Optimizations that we used to generate these variants include loop unrolling, cache/register tiling and scalar replacement. Each of these kernels and kernel-variants is configured to run with multiple working set sizes and each computation is configured to run for at least 2 seconds. With the computational kernels and kernel variants configured to run on multiple working set sizes, we had a total of 538 computations that formed our empirical dataset.

Each computation in the aforementioned dataset is timed on the test system under different DRAM and CPU power caps. We start by running the benchmark computations with no power bound (base case) and denote the average power measurements from RAPL for CPU and DRAM domains as P_{bCPU} and P_{bDRAM}. For each computation, we then generate random DRAM power caps that are in $[0.7 \times P_{bDRAM}, P_{bDRAM}]$ set (i.e., we select random configurations where we reduce the DRAM power cap by up to 30%.). The same is also done to generate CPU power cap levels. A simple cartesian product combines the DRAM and CPU candidate power caps to form a set of simultaneous DRAM and CPU power cap configurations. Of those, we then select 18 random configurations[3] and run the computation on those 18 configurations. We execute each of the configurations six times to get six power and performance measurements; we discard the minimum and maximum measurements and average the remaining four. Also, for each computation we generate a characterization signature using the tools described in Section III-B. After post-processing, we have a total of 9550 data points in our empirically collected dataset.

C. Model Formulation, Results and Diagnostics

Equation 2 shows a more specific formulation of Equation 1. P_d is the performance degradation that power capping induces. $cpu\_reduction$ and $dram\_reduction$ are CPU and DRAM power bound indicators as percentage reductions with respect to power drawn by respective domains when there are no power caps. $pi\_v[0,1,2]$ are L1, L2 and L3 hits per instruction, while pi_v3m measures accesses that miss in L3 (and therefore go to main memory) per instruction. $fprat$ is the floating point ratio (floating point operations per memory operation) and $bytespermemop$ is the bytes transferred per memory access. ins_mix consists of instruction mix parameters – branch ops, load ops, store ops etc. expressed as a percentage of the total dynamic instructions. idu and fdu are integer floating point data dependence metrics.

$$P_d = f(dram\_reduction, cpu\_reduction, pi\_v0m, pi\_v1m, pi\_v2m, pi\_v3m, fprat, ins\_mix, bytespermemop, idu, fdu) \quad (2)$$

To generate and evaluate P_d models (see Equation 2), we first perform a 60%-40% split on our overall empirical data. The first 60% portion is used to train the model using 10-fold cross-validation. The remaining 40% data is used to test the accuracy of the model. Model evaluation results are shown in Figure 3. The Figure shows the modeled versus measured runtime slowdowns (with respect to no power capping) for both the training and the test data. For a well-behaved and accurate model, the points in the graph should be roughly clustered around the diagonal 45 degree line, which is the case in Figure 3. The absolute mean error in predicting the test dataset is less than 2%.

We calculate the relative variable importance for each of the input predictors in Equation 2 to identify dominant

[3]We chose to run 18 configurations based on our experience with the time it takes to generate training set samples. We wanted to minimize the number of training samples, but at the same time generate enough points for our model training to succeed.

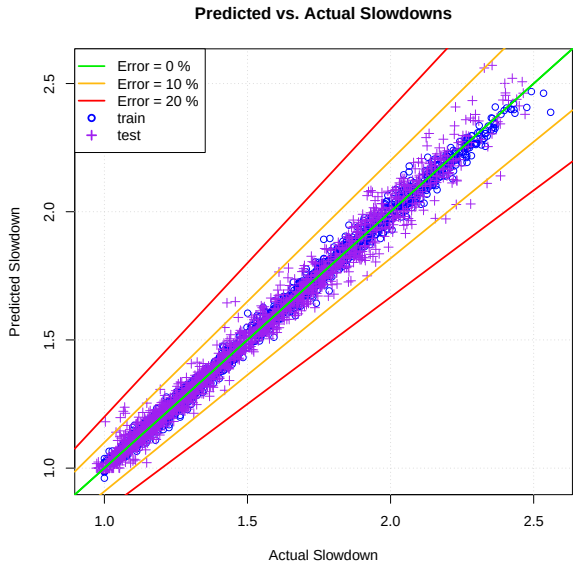

Fig. 3. Model diagnostics: The figure shows the prediction accuracy of the models for both the training and test sets formed by splitting the overall empirical data into 60%-40% parts.

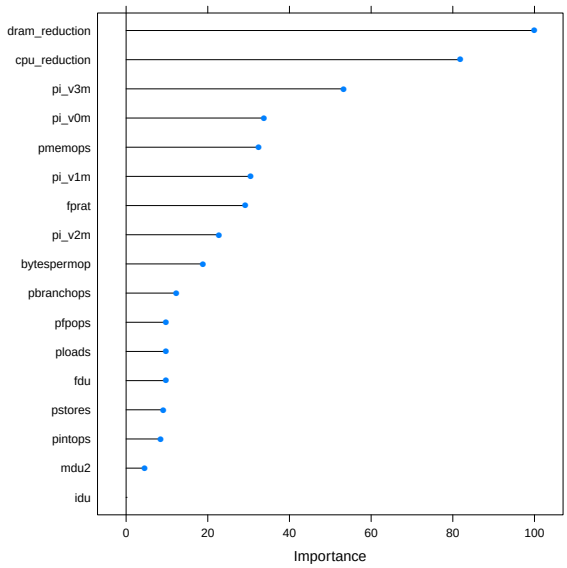

Fig. 4. Model diagnostics: Variable importance plot, which shows the ranking of the predictors used to train the model.

inputs as well as those inputs that have minimal effect on the performance degradation and could potentially be dropped from the model. Figure 4 presents the variable importance. As expected, CPU and DRAM power cap levels have the most impact, followed by metrics that quantify computations' interaction with the memory hierarchy. Finally, instruction mix parameters round out the top predictors. From among the elements in the computation signature, the trips to main memory per instruction (pi_v3m) is the most important, which is to be expected since more trips to main memory would mean more CPU stalls and therefore less sensitivity to the CPU power cap reduction (and the reverse phenomenon for DRAM power cap reduction).

D. Model-based Auto-tuning

One of the use-cases of the models that we have developed in this work is its easy applicability within a search-based auto-tuning workflow, where multiple variants of a given code that use different compiler-based optimizations are generated and evaluated to find the one that performs the best [23]. In a dynamic auto-tuning environment, which has the capability to respond to the changes in the operating environment, our models can be used to quickly identify the variant that performs the best within a power budget. We make a reasonable assumption that we have access to the performance of all the variants evaluated during the auto-tuning process in base setting with no power caps. Using that data and the stipulated power budget, our models can inform the execution times of the variants on all possible configurations of simultaneous DRAM and CPU power caps, thereby making the selection of the best performing variant possible.

To illustrate, we use the ATAX (matrix transpose and vector multiplication) kernel. The empirical dataset that we use to train and test the models consists of 100 ATAX kernel variants generated using the Orio [16] tool. Based on the performance at the base settings of no power capping, we selected the top-10 variants. Each of these selected variants was then evaluated by imposing 100 randomly selected DRAM and CPU power bounds. The key question that we want to answer with this set of experiments is the following – how will reducing the overall DRAM and CPU power bound affect the choice of the optimal variant? That is, will the best performing variant continue to remain the best at a reduced power cap? If we assume a scenario where there is a need to reduce the overall power cap by 10%, will it make sense to run a different variant. Note that the 10% reduction will be based on the DRAM+CPU power drawn by the best performing variant at base setting.

In Figure 5 we show the results of this investigation. The X-axis is total power reduction as a percentage of the power drawn by best variant (variant v87) at no power cap; the Y-axis shows the range of execution

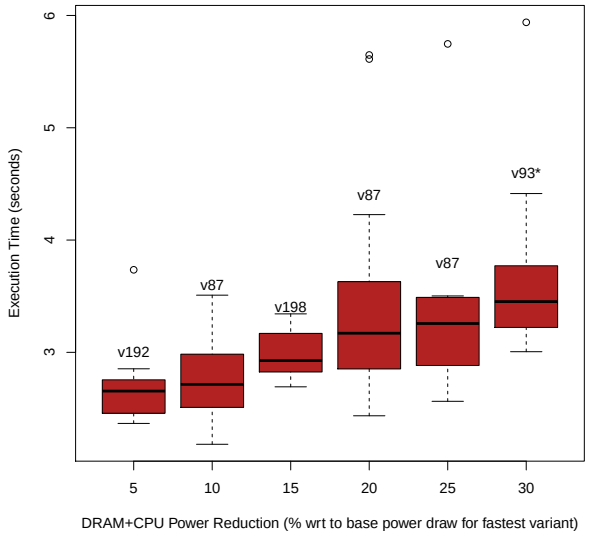

Fig. 5. Model evaluation on additional ATAX variants to show how our model could be used within the auto-tuning workflow.

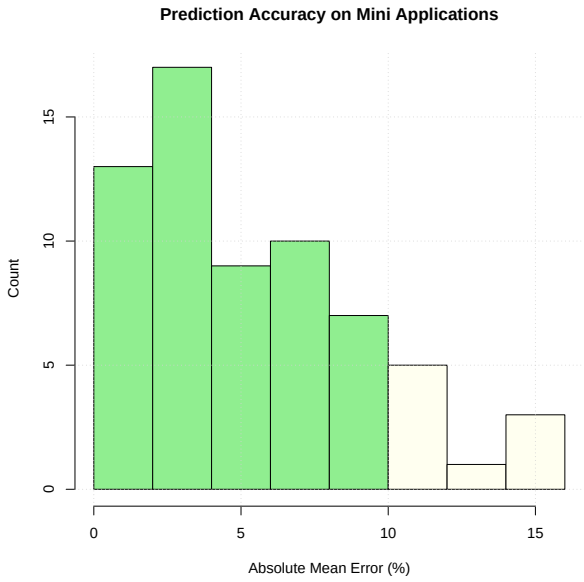

Fig. 6. Evaluation of the model on mini applications – miniGhost and CoMD. More than 86% of the cases have less than 10% error.

times (in seconds) of the selected top-10 variants. At 5% reduction in power draw, v192 should be selected, which is indicated in the graph. When there is a star placed next to the variant, we are indicating that our model did not agree (i.e., the variant selected using the exec time predicted by our model did not match the variant selected using the real execution time and that happens in only one out of six cases). The answer to the research question that we posed is that the ranking of variants in terms of execution time does change at different power caps. The objective of this exercise was to show that given a set of power allocation schemes for DRAM and CPU, our model is able to predict the performance degradation of any available variant.

E. Model Evaluation on Mini Applications

Finally, we illustrate the model's capability in not just kernels, but utilize the models to predict the behavior of two mini applications in a power capped operating environments – miniGhost [2] and CoMD [11]. miniGhost is a Finite Difference mini-application which implements a difference stencil across a homogenous 3D domain. We consider 128^3 and 256^3 sized grids for our evaluations. CoMD is a proxy application for a broad class of molecular dynamics simulations and provides implementations for calculating simple Lennard-Jones (LJ) and Embedded Atom Method (EAM) potentials. We consider both in our experiments.

Using our application characterization tool-suite, we isolate the key hotspots in these applications and feed the characterization to our model to ask for performance degradation predictions for a set of randomly generated CPU and DRAM power cap configurations. To evaluate the model accuracy we measure the real performance degradations for those configurations by running the application under given CPU and DRAM power limits.

The predicted degradations and actual degradations are then compared to validate the model's prediction accuracy. Overall prediction results (histogram) are shown in Figure 6. The error metric reported here are for 'out of sample' tests, i.e., the characterization signatures for the application hotspots are not seen during the model training process and thereby demonstrate the predictive accuracy of our models on computations in HPC applications. Overall the models predict the outcome well – average absolute mean error is 6%. For more than 86% of the application hotspots, the prediction error is less than 10%.

V. Conclusions

Performance implications of power capping are largely determined by the computational characteristics of an application, i.e., different computations will have a different performance behavior when run in power-capped environments. This paper presented a highly accurate model-based methodology that utilizes application characteristics to inform the selection of performance max-

imizing power cap configurations for the CPU and DRAM domains.

ACKNOWLEDGEMENTS

This work was supported in part by the DOE Office of Science, Advanced Scientific Computing Research, under award number 62855 "Beyond the Standard Model – Towards an Integrated Modeling Methodology for the Performance and Power"; PNNL lead institution; Program Manager Karen Pao. The authors acknowledge partial support from LLNL under subcontract B600667. This work was also supported in part by the DoD and used elements at the Extreme Scale Systems Center, located at ORNL and funded by the DoD. Partial support also came from the DOE Office of Science through the SciDAC award titled SUPER (Institute for Sustained Performance, Energy and Resilience). Part of this work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

REFERENCES

[1] Intel™Power Governor. https://software.intel.com/en-us/articles/intel-power-governor.

[2] Mantevo Project. http://mantevo.org/.

[3] Advanced Mico Devices. Bios and kernel developer's guide (bkdg) for amd family 15h models 00h-0fh processors. Jan. 2012.

[4] P. Balaprakash, S. M. Wild, and B. Norris. SPAPT: Search problems in automatic performance tuning. *Procedia Computer Science*, 9:1959–1968, 2012.

[5] B. Behle, N. Bofferding, M. Broyles, C. Eide, M. Floyd, C. Francois, A. Geissler, M. Hollinger, H.-Y. McCreary, and C. Rath. Ibm energyscale for power6 processor-based systems. 2009.

[6] M. Broyles, C. Francois, A. Geissler, G. Grout, M. Hollinger, T. Rosedahl, G. Silva, M. Vanderwiel, J. Van Heuklon, and B. Veale. Ibm energyscale for power7 processor-based systems. 2010.

[7] C. Chen, J. Chame, and M. W. Hall. CHiLL: A framework for composing high-level loop transformations. TR 08-897, Univ. of Southern California, Jun 2008.

[8] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. Rapl: Memory power estimation and capping. In *Low-Power Electronics and Design (ISLPED), 2010 ACM/IEEE International Symposium on*, pages 189–194, Aug 2010.

[9] H. Esmaeilzadeh, E. Blem, R. St Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376. IEEE, 2011.

[10] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Parallel job scheduling for power constrained hpc systems. *Parallel Computing*, 38(12):615–630, 2012.

[11] ExMatEx. CoMD Proxy Application. http://www.exmatex.org/comd.html, 2012. Online; accessed 18-January-2015.

[12] K. Fukazawa, M. Ueda, M. Aoyagi, T. Tsuhata, K. Yoshida, A. Uehara, M. Kuze, Y. Inadomi, and K. Inoue. Power consumption evaluation of an mhd simulation with cpu power capping. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 612–617, May 2014.

[13] B. Goel, S. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati. Portable, scalable, per-core power estimation for intelligent resource management. In *Green Computing Conference, 2010 International*, pages 135–146, Aug 2010.

[14] M. Laurenzano, M. Tikir, L. Carrington, and A. Snavely. Pebil: Efficient static binary instrumentation for linux. In *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, pages 175 –183, march 2010.

[15] D. Li, B. de Supinski, M. Schulz, D. Nikolopoulos, and K. Cameron. Strategies for energy-efficient resource management of hybrid programming models. *Parallel and Distributed Systems, IEEE Transactions on*, 24(1):144–157, Jan 2013.

[16] B. Norris, A. Hartono, and W. Gropp. Annotations for productivity and performance portability. In *Petascale Computing: Algorithms and Applications*, Computational Science, pages 443–462. Chapman & Hall / CRC Press, 2007.

[17] T. Patki, D. K. Lowenthal, B. Rountree, M. Schulz, and B. R. de Supinski. Exploring hardware overprovisioning in power-constrained, high performance computing. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 173–182, New York, NY, USA, 2013. ACM.

[18] L. Porter, M. A. Laurenzano, A. Tiwari, A. Jundt, W. A. Ward, Jr., R. Campbell, and L. Carrington. Making the most of smt in hpc: System- and application-level perspectives. *ACM Trans. Archit. Code Optim.*, 11(4):59:1–59:26, Jan. 2015.

[19] L.-N. Pouchet. *PolyBench: The Polyhedral Benchmark suite*, 2012. http://www.cse.ohio-state.edu/~pouchet/software/polybench/.

[20] B. Rountree, D. Ahn, B. de Supinski, D. Lowenthal, and M. Schulz. Beyond dvfs: A first look at performance under a hardware-enforced power bound. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 947–953, May 2012.

[21] RuleQuest Research. Data Mining with Cubist. http://rulequest.com/cubist-info.html, 2012. Online; accessed 18-January-2015.

[22] O. Sarood, A. Langer, L. Kale, B. Rountree, and B. de Supinski. Optimizing power allocation to cpu and memory subsystems in overprovisioned hpc systems. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pages 1–8, Sept 2013.

[23] A. Tiwari, C. Chen, J. Chame, M. Hall, and J. Hollingsworth. A scalable auto-tuning framework for compiler optimization. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–12, May 2009.

[24] A. Tiwari, A. Gamst, M. Laurenzano, M. Schulz, and L. Carrington. Modeling the impact of reduced memory bandwidth on hpc applications. In F. Silva, I. Dutra, and V. Santos Costa, editors, *Euro-Par 2014 Parallel Processing*, volume 8632 of *Lecture Notes in Computer Science*, pages 63–74. Springer International Publishing, 2014.

[25] A. Tiwari, M. Laurenzano, L. Carrington, and A. Snavely. Modeling power and energy usage of hpc kernels. In *Proceedings of the Eighth Workshop on High-Performance, Power-Aware Computing 2012*, HPPAC '12, 2012.

[26] A. Tiwari, J. Peraza, M. Laurenzano, L. Carrington, and A. Snavely. Green queue: Customized large-scale clock fre-

quency scaling. In *Proceedings of the Second International Conference on Cloud and Green Computing*, CGC '12, 2012.